Graph-based, Self-Supervised Program Repair from Diagnostic Feedback

ICML 2020

Michihiro Yasunaga, Percy Liang
Stanford University
Why program repair?

Programmers spend 75% of time fixing source code errors

Automatic program repair can dramatically enhance programming productivity
Theme: Learning from Feedback

Humans learn from feedback
Write code → compile/execute → repair code based on feedback
Theme: Learning from Feedback

Humans learn from feedback

Write code → compile/execute → repair code based on feedback
Theme: Learning from Feedback

Humans learn from feedback
Write code $\rightarrow$ compile/execute $\rightarrow$ repair code based on feedback

General framework with many other applications
- Edit essays based on written feedback
- Learn from user inputs in interactive dialogue

Repair programs based on errors
Edit essays based on feedback
```
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++) {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++) {
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    }
    ...
```
```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++){
        tmp.push_back( a[i] );
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++){
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    } ...
```

Should be `string`
Example

**Broken program**

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string,int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++){
        tmp.push_back( a[i] );
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++){
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    } ...
```

**Evaluator (compiler)**

**Feedback**

*line 9:error:* request for member ‘size’ in ‘a’, which is of non-class type ‘char’
Example

Broken program

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++) {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++) {
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    }
    ...
```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main()
{
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++)
    {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++)
        {
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    }
    ...
```
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++) {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++) {
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    }
    ...
Example

Broken program

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string,int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++)
        tmp.push_back(a[i]);
    string tmp1 = tmp;
    for (j = 0; j < b.size(); j++)
        tmp1.push_back(b[j]);
    mp[tmp1] = 1;
}
...}
```

Feedback

line 9: error: request for member ‘size’ in ‘a’, which is of non-class type ‘char’
Example

Broken program

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++) {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++) {
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    }
    ...}
```

Evaluator (compiler)

Feedback

line 9: error: request for member ‘size’ in ‘a’, which is of non-class type ‘char’

Repair!

1. Error localized line 5
2. Repair
   ```cpp
   char tmp, a, b;
   → string tmp, a, b;
   ```
Goal: learn to repair programs from error messages
Goal: learn to repair programs from error messages

### Broken program

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string,int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++) {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++) {
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    }
    ...
}
```

### Diagnostic Feedback

**line 9:** error: request for member ‘size’ in ‘a’, which is of non-class type ‘char’
Goal: learn to repair programs from error messages

Broken program

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++) {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++) {
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    }
    ...}
```

Diagnostic Feedback

- **line 9:** error: request for member ‘size’ in ‘a’, which is of non-class type ‘char’

**DrRepair (our system)**

1. Error localized **line 5**
Goal: learn to repair programs from error messages

Broken program

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;

int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++){
        tmp.push_back(a[i]);
        string tmp1 = tmp;
        for (j = 0; j < b.size(); j++){
            tmp1.push_back(b[j]);
            mp[tmp1] = 1;
        }
    } ...
```

Diagnostic Feedback

- **line 9:** error: request for member ‘size’ in ‘a’, which is of non-class type ‘char’

**DrRepair** (our system)

1. Error localized **line 5**
2. Repair

```cpp
char tmp, a, b;
→ string tmp, a, b;
```
Challenges

1. Modeling
   - How to connect two modalities: **program** and **feedback**?
   - How to model the **reasoning** of repair (e.g. tracking symbols)?

```cpp
int main() {
    char tmp, a, b;
    map<string,int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++)
        tmp.push_back(a[i]);
    string tmp1 = tmp;
    ...
```

Compiler message
request for member ‘size’ in ‘a’, which is of non-class type ‘char’
Challenges

1. Modeling
   o How to connect two modalities: program and feedback?
   o How to model the reasoning of repair (e.g. tracking symbols)?

2. Data
   o Existing works rely on labeled datasets of <broken code, fixed code>
   o Relatively small (10–100K data points). How to scale up?
Our contributions

1. Program-feedback graph
   ○ Connect symbols across **program** & **feedback**
   ○ Performs reasoning via **graph-attention**

Source code

```
4 int main() {
5   char tmp, a, b;
6   map<string,int> mp;
7   cin >> a >> b;
8   int i, j;
9   for (i = 0; i < a.size(); ...
10  tmp.push_back(a[i]);
11  string tmp1 = tmp;
...```

Compiler message

`request for member 'size' in 'a', which is of non-class type 'char'`
Our contributions

2. Self-supervised learning
   ○ Collect unlabeled programs
   ○ Corrupt and get diagnostic feedback (e.g. run compiler)
   ⇒ Extra training data: <broken code, feedback, fixed code>

Working code

5   int i, n;
6   string A;
7   cin >> n;
8   A.resize(n);
9   for (i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }

Corrupted

5   int i, n;
6   char A;
7   cin >> n;
8   A.resize(n);
9   for (i=0;i<n;i++){
10   cin >> A[j];
11   cout << i; }

Error!
   line 7: expected ‘;’

Error!
   line 7: expected ‘;’
Our results

Improved performance on two applications
- DeepFix: correct intro programming assignments in C
- SPoC: correct output of C++ program synthesis

DeepFix Test

<table>
<thead>
<tr>
<th>Method</th>
<th>No Compiler</th>
<th>Base</th>
<th>Base + Graph</th>
<th>Ours (DrRepair)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gupta+17</td>
<td>27.0%</td>
<td>34.0%</td>
<td>66.4%</td>
<td>68.2%</td>
</tr>
<tr>
<td>Gupta+19</td>
<td>26.6%</td>
<td>34.0%</td>
<td>66.4%</td>
<td></td>
</tr>
<tr>
<td>Hajipour+19</td>
<td>45.3%</td>
<td>62.5%</td>
<td>68.2%</td>
<td></td>
</tr>
</tbody>
</table>

SPoC TestP

<table>
<thead>
<tr>
<th>Method</th>
<th>Top1 stitch</th>
<th>Best first search</th>
<th>Kulal+19</th>
<th>Ours (DrRepair)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>17.8%</td>
<td>32.5%</td>
<td>34.2%</td>
<td>37.6%</td>
</tr>
</tbody>
</table>
Outline

● Innovations
  1. Reasoning via program-feedback graph
  2. Self-supervised learning

● Evaluations
  1. DeepFix
  2. SPoC

● Analysis & Examples

● Takeaways
1. Reasoning via program-feedback graph
1. Reasoning via program-feedback graph

Challenges

- How to connect two modalities: program and feedback?
- How to model the reasoning of repair (e.g. tracking symbols)?

Source code

```cpp
int main() {
  char tmp, a, b;
  map<string,int> mp;
  cin >> a >> b;
  int i, j;
  for (i = 0; i < a.size() ...)
    tmp.push_back(a [i]);
  string tmp1 = tmp;
}
```

Compiler message

request for
member ‘size’ in
‘a’, which is of
non-class type
‘char’
1. Reasoning via program-feedback graph

Our solution: **program-feedback graph**

Source code:

```c++
int main() {
    char tmp, a, b;
    map<string,int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); ...)
        tmp.push_back(a[i]);
    string tmp1 = tmp;
}
```

Compiler message:

Request for member `size` in `a`, which is of non-class type `char`.
1. Reasoning via program-feedback graph

Our solution: **program-feedback graph**
- Join program & feedback through symbols relevant to program repair
  → shared/abstracted semantic space

Source code

```cpp
4 int main() {
5  char tmp, a, b;
6  map<string,int> mp;
7  cin >> a >> b;
8  int i, j;
9  for (i = 0; i < a.size(); ...)
10    tmp.push_back(a[i]);
11  string tmp1 = tmp;
```
1. Reasoning via program-feedback graph

Our solution: **program-feedback graph**

- Join program & feedback through symbols relevant to program repair → **shared/abstracted semantic space**
- Reason over this space using **graph attention**

**Source code**

```
4 int main() {
5   char tmp, a, b;
6   map<string,int> mp;
7   cin >> a >> b;
8   int i, j;
9   for (i = 0; i < a.size(); ...)
10   tmp.push_back(a[i]);
11   string tmp1 = tmp;
```
1. Reasoning via program-feedback graph

How to construct graph?
- Recognize token types

Source code
```cpp
int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); ...) {
        tmp.push_back(a[i]);
        string tmp1 = tmp;
    }
```

Compiler message
request for member ‘size’ in ‘char’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types

Source code

```c++
4 int main() {
5  char tmp, a, b;
6  map<string,int> mp;
7  cin >> a >> b;
8  int i, j;
9  for (i = 0; i < a.size() ...
10   tmp.push_back(a [i]);
11   string tmp1 = tmp;
```

Compiler message

request for member ‘size’ in ‘a’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

How to construct graph?
- Recognize token types

```cpp
4 int main() { 
5  char tmp, a, b;
6  map<string,int> mp;
7  cin >> a  >>  b;
8  int i, j;
9  for (i = 0; i < a.size(); ... 
10   tmp.push_back(a [i]);
11  string tmp1 = tmp;
```

Compiler message
- request for member ‘size’ in ‘a’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types

```cpp
4 int main() {
5  char tmp, a, b;
6  map<string,int> mp;
7  cin >> a >> b;
8  int i, j;
9  for (i = 0; i < a.size() ... 
10   tmp.push_back(a[i]);
11   string tmp1 = tmp;
```

Compiler message

request for

member ‘size’ in

‘a’, which is of

non-class type

‘char’
1. Reasoning via program-feedback graph

How to construct graph?
- Recognize token types

```cpp
int main() {
  char tmp, a, b;
  map<string, int> mp;
  cin >> a >> b;
  int i, j;
  for (i = 0; i < a.size(); ...) {
    tmp.push_back(a[i]);
  }
  string tmp1 = tmp;
}
```

Compiler message
request for
member ‘size’ in
‘a’, which is of
non-class type
‘char’
1. Reasoning via program-feedback graph

How to construct graph?
- Recognize token types

Source code:
```c++
4 int main() {
5   char tmp, a, b;
6   map<string,int> mp;
7   cin >> a >> b;
8   int i, j;
9   for (i = 0; i < a.size() ...)
10   tmp.push_back(a[i]);
11   string tmp1 = tmp;
```

Compiler message:
request for
member ‘size’ in
‘a’, which is of
non-class type
‘char’
1. Reasoning via program-feedback graph

How to construct graph?
- Recognize token types

```cpp
int main() {
  char tmp, a, b;
  map<string, int> mp;
  cin >> a >> b;
  int i, j;
  for (i = 0; i < a.size(); ...)
    tmp.push_back(a[i]);
  string tmp1 = tmp;
}
```

Compiler message
- request for member ‘size’ in ‘a’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types

```cpp
int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); ...)
        tmp.push_back(a[i]);
    string tmp1 = tmp;
}
```

Compiler message:

request for member ‘size’ in ‘a’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types

Source code:

```cpp
4 int main() {
5  char tmp, a, b;
6  map<string, int> mp;
7  cin >> a >> b;
8  int i, j;
9  for (i = 0; i < a.size() ...) ... 
10  tmp.push_back(a[i]);
11  string tmp1 = tmp;
```

Compiler message:

request for member ‘size’ in ‘a’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

How to construct graph?

● Recognize token types
● **Nodes**: diagnostic arguments

---

**Source code**

```cpp
4 int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size() ... 
10    tmp.push_back(a [i]);
11    string tmp1 = tmp;
```
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types
- **Nodes:** diagnostic arguments, their occurrences

```cpp
int main() {
  char tmp, a, b;
  map<string, int> mp;
  cin >> a >> b;
  int i, j;
  for (i = 0; i < a.size(); ...)
    tmp.push_back(a[i]);
  string tmp1 = tmp;
  // Compiler message
  // request for member 'size' in 'a', which is of non-class type 'char'
```
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types
- **Nodes**: diagnostic arguments, their occurrences

---

**Source code**

```cpp
4 int main() {
5    char tmp, a, b;
6    map<string, int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); i++) ... 
10   tmp.push_back(a[i]);
11   string tmp1 = tmp;
```
1. Reasoning via program-feedback graph

How to construct graph?
- Recognize token types
- **Nodes**: diagnostic arguments, their occurrences

```cpp
4 int main() {
5    char tmp, a, b;
6    map<string,int> mp;
7    cin >> a >> b;
8    int i, j;
9    for (i = 0; i < a.size(); ...)
10       tmp.push_back(a[i]);
11    string tmp1 = tmp;
```
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types
- **Nodes**: diagnostic arguments, their occurrences, and all identifiers

```cpp
int main() {
  char tmp, a, b;
  map<string, int> mp;
  cin >> a >> b;
  int i, j;
  for (i = 0; i < a.size(); ...)
    tmp.push_back(a[i]);
  string tmp1 = tmp;
}
```

Compiler message
- request for member ‘size’ in ‘a’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

How to construct graph?

- Recognize token types
- **Nodes**: diagnostic arguments, their occurrences, and all identifiers
- **Edges**: connect identical tokens to capture *semantic correspondence*

```
int main() {
  char tmp, a, b;
  map<string, int> mp;
  cin >> a >> b;
  int i, j;
  for (i = 0; i < a.size(); ...)
    tmp.push_back(a[i]);
  string tmp1 = tmp;
  ...}
```

*Compiler message*

request for member ‘size’ in ‘a’, which is of non-class type ‘char’
1. Reasoning via program-feedback graph

Model
- Initial encoding
- Graph attention
- Recontextualization
- Decoding
1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member
‘size’...
1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
... 

Compiler message
9: request for member
‘size’ ...

Line 1
1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member
   ‘size’ ...

Line 1 Line 2
1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member
‘size’ ...

Line 1  Line 2  Line 3
Source code
1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member ‘size’ ...

Line 1  Line 2  Line 3
Source code

Line idx  Msg content
Feedback (compiler message)
1. Reasoning via program-feedback graph

Model (Initial encoding)

Source code
1 int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...
1. Reasoning via program-feedback graph

Model (Graph attention)

Source code
1 int main() {
2 char tmp, a, b;
3 map<string,int> mp;
...

Compiler message
9: request for member ‘size’ …
1. Reasoning via program-feedback graph

Model (Graph attention)
- Message passing across tokens with long-range dependencies

Source code
1 int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member ‘size’ ...

Program-Feedback Graph
1. Reasoning via program-feedback graph

Model (Graph attention)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member ‘size’ …
1. Reasoning via program-feedback graph

Model (Recontextualization)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member ‘size’ …

Graph Attention

Line 1   :  Line 2   :  Line 3   Line idx : Msg content
1. Reasoning via program-feedback graph

Model (Recontextualization)

Source code
1 int main() {
2 char tmp, a, b;
3 map<string,int> mp;
...}

Compiler message
9: request for member ‘size’ ...

Graph Attention

Line 1 : Line 2 : Line 3
Line idx : Msg content
1. Reasoning via program-feedback graph

Model (Recontextualization)

Source code
1  int main() {
2  char tmp, a, b;
3  map<string,int> mp;
...

Compiler message
9: request for member ‘size’ ...

1. Reasoning via program-feedback graph
Model (Recontextualization)
1. Reasoning via program-feedback graph

Model (Decoding)

Source code
1 int main() {
2 char tmp, a, b;
3 map<string,int> mp;
...

Compiler message
9: request for member ‘size’ ...

Graph Attention

1. Reasoning via program-feedback graph

Model (Decoding)

Source code
1  int main() { 
2    char tmp, a, b; 
3    map<string,int> mp; 
... 

Compiler message
9: request for member ‘size’ ...

Graph Attention

Line 1 : Line 2 : Line 3
Line idx : Msg content

Localize = 2   Repair = "string tmp,a,b;"

LSTM Code (2)   LSTM Code (2)   LSTM Code (2)   LSTM Code (2)

MLP + softmax

Pointer-Generator Decoder

LSTM Code (3)

Graph Attention

MST
1. Reasoning via program-feedback graph

Model overview
2. Self-supervised learning
2. Self-supervised learning

Why?
- Labeled datasets of program repair are small (10-100K examples)
- Vast amount of **unlabeled programs** available online
- Can we leverage them to improve learning?

GitHub

> 30M repos

Codeforces

>> 1M submissions
2. Self-supervised learning

Our idea (outline)

Step 1. Collect unlabeled, working programs \( y \)

Step 2. Design \((\text{randomized})\) program corruption procedure \( P \)

Step 3. Corrupt and get diagnostic feedback (e.g. run compiler)

\[ \Rightarrow \text{Extra training data: } \langle \text{broken code } x, \text{ feedback } f, \text{ fixed code } y \rangle \]

Step 4. Use them for pre-training
2. Self-supervised learning

1. Collect unlabeled programs
   - Our target tasks (DeepFix & SPoC) are in C/C++
   - Collect 300K working C++ programs from codeforces.com
2. Self-supervised learning

2. How to design corruption procedure $P$?

- Look at common errors (know your enemy)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Common compiler messages</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected ...</td>
<td>expected @@ (e.g. expected ‘;’ before...)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>missing @@ (e.g. missing &quot; )</td>
<td></td>
</tr>
<tr>
<td>operator/punctuator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>primary expression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifier <strong>type</strong></td>
<td>redeclaration/conflicting declaration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid conversion from &lt;type&gt; to &lt;type&gt;</td>
<td></td>
</tr>
<tr>
<td>Identifier <strong>undeclared</strong></td>
<td>@@@ was not declared</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>‘else’ without a previous ‘if’</td>
<td></td>
</tr>
<tr>
<td></td>
<td>no matching function for call to...</td>
<td></td>
</tr>
</tbody>
</table>

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]
2. Self-supervised learning

2. How to design corruption procedure $P$?
- Look at common errors (know your enemy)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Common compiler messages</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected ...</td>
<td>expected @@ (e.g. expected ‘;’ before...)</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>missing @@ (e.g. missing ”)</td>
<td></td>
</tr>
<tr>
<td>operator/punctuator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>primary expression</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifier type</td>
<td>redeclaration/conflicting declaration</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>invalid conversion from &lt;type&gt; to &lt;type&gt;</td>
<td></td>
</tr>
<tr>
<td>Identifier undeclared</td>
<td>@@@ was not declared</td>
<td>62%</td>
</tr>
<tr>
<td>Others</td>
<td>‘else’ without a previous ‘if’</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>no matching function for call to...</td>
<td></td>
</tr>
</tbody>
</table>

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]
2. Self-supervised learning

2. How to design corruption procedure $P$?
   - Look at common errors (know your enemy)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Common compiler messages</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected ...</td>
<td>expected @@ (e.g. expected ‘;’ before...)</td>
<td>Experienced: 9%</td>
</tr>
<tr>
<td></td>
<td>missing @@ (e.g. missing &quot; )</td>
<td>Beginner: 48%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37%</td>
</tr>
<tr>
<td>Identifier type</td>
<td>redeclaration/conflicting declaration</td>
<td>Experienced: 9%</td>
</tr>
<tr>
<td></td>
<td>invalid conversion from &lt;type&gt; to &lt;type&gt;</td>
<td>Beginner: 5%</td>
</tr>
<tr>
<td>Identifier undeclared</td>
<td>@@@ was not declared</td>
<td>Experienced: 62%</td>
</tr>
<tr>
<td>Others</td>
<td>‘else’ without a previous ‘if’</td>
<td>Beginner: 33%</td>
</tr>
<tr>
<td></td>
<td>no matching function for call to...</td>
<td>20%</td>
</tr>
</tbody>
</table>

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]
2. Self-supervised learning

2. How to design corruption procedure $P$?
- Look at common errors (know your enemy)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Common compiler messages</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected ...</td>
<td></td>
<td>Experienced</td>
</tr>
<tr>
<td>- operator/punctuator</td>
<td>expected @@ (e.g. expected ‘;’ before...)</td>
<td>9%</td>
</tr>
<tr>
<td>- primary expression</td>
<td>missing @@ (e.g. missing &quot;&quot;)</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11%</td>
</tr>
<tr>
<td>Identifier type</td>
<td>redeclaration/conflicting declaration</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>invalid conversion from &lt;type&gt; to &lt;type&gt;</td>
<td>9%</td>
</tr>
<tr>
<td>Identifier undeclared</td>
<td>@@ was not declared</td>
<td>62%</td>
</tr>
<tr>
<td>Others</td>
<td>‘else’ without a previous ‘if’</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>no matching function for call to...</td>
<td>20%</td>
</tr>
</tbody>
</table>

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]
2. Self-supervised learning

2. How to design corruption procedure $P$?
   - Look at common errors (know your enemy)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Common compiler messages</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expected ...</strong></td>
<td>expected @@ (e.g. expected ‘;’ before...)</td>
<td>9%</td>
</tr>
<tr>
<td>• operator/punctuator</td>
<td>missing @@ (e.g. missing &quot; )</td>
<td>48%</td>
</tr>
<tr>
<td>• primary expression</td>
<td></td>
<td>35%</td>
</tr>
<tr>
<td><strong>Identifier type</strong></td>
<td>redeclaration/conflicting declaration</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>invalid conversion from $&lt;$type$&gt;$ to $&lt;$type$&gt;$</td>
<td>5%</td>
</tr>
<tr>
<td><strong>Identifier undeclared</strong></td>
<td>@@@ was not declared</td>
<td>18%</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td>‘else’ without a previous ‘if’</td>
<td>62%</td>
</tr>
<tr>
<td></td>
<td>no matching function for call to...</td>
<td>33%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>31%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Experienced</th>
<th>Beginner</th>
<th>SPoC</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expected ...</strong></td>
<td>9%</td>
<td>48%</td>
<td>35%</td>
<td>30%</td>
</tr>
<tr>
<td>• operator/punctuator</td>
<td></td>
<td>37%</td>
<td>29%</td>
<td>23%</td>
</tr>
<tr>
<td>• primary expression</td>
<td></td>
<td>11%</td>
<td>6%</td>
<td>7%</td>
</tr>
<tr>
<td><strong>Identifier type</strong></td>
<td>9%</td>
<td>5%</td>
<td>18%</td>
<td>11%</td>
</tr>
<tr>
<td><strong>Identifier undeclared</strong></td>
<td>62%</td>
<td>33%</td>
<td>31%</td>
<td>42%</td>
</tr>
<tr>
<td><strong>Others</strong></td>
<td>20%</td>
<td>14%</td>
<td>16%</td>
<td>17%</td>
</tr>
</tbody>
</table>

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]
2. Self-supervised learning

2. How to design corruption procedure $P$?
   - Look at common errors (know your enemy)

<table>
<thead>
<tr>
<th>Error type</th>
<th>Common compiler messages</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected ...</td>
<td>expected @@ (e.g. expected ‘;’ before...)</td>
<td>Experienced</td>
</tr>
<tr>
<td></td>
<td>missing @@ (e.g. missing &quot; )</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6%</td>
</tr>
<tr>
<td>Identifier type</td>
<td>redeclaration/conflicting declaration</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>invalid conversion from &lt;type&gt; to &lt;type&gt;</td>
<td></td>
</tr>
<tr>
<td>Identifier undeclared</td>
<td>@@@ was not declared</td>
<td>62%</td>
</tr>
<tr>
<td>Others</td>
<td>‘else’ without a previous ‘if’</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>no matching function for call to...</td>
<td></td>
</tr>
</tbody>
</table>

[Gupta et al. 17] [Mesbah et al. 19] [Kulal et al. 19]
2. Self-supervised learning

2. How to design corruption procedure $P$?
   - Look at common errors
   - Design perturbation modules $M$ to cause those errors
2. Self-supervised learning

2. How to design corruption procedure $P$?
   - Look at common errors
   - Design perturbation modules $M$ to cause those errors

<table>
<thead>
<tr>
<th>Perturbation module</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax (delete/insert/replace operators .;(){}}{{&quot;+}}, etc.)</td>
<td>return 0; } → return 0; }</td>
</tr>
</tbody>
</table>
2. Self-supervised learning

2. How to design corruption procedure $P$?
- Look at common errors
- Design perturbation modules $M$ to cause those errors

<table>
<thead>
<tr>
<th>Perturbation module</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax (delete/insert/replace operators .;(){}''+, etc.)</td>
<td>return 0; } → return 0; } }</td>
</tr>
<tr>
<td>ID-type (delete/insert/replace type)</td>
<td>string tmp; → char tmp;</td>
</tr>
</tbody>
</table>
2. Self-supervised learning

2. How to design corruption procedure $P$?
   - Look at common errors
   - Design perturbation modules $M$ to cause those errors

<table>
<thead>
<tr>
<th>Perturbation module</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong> <em>(delete/insert/replace operators .;(){'}&quot;+, etc.)</em></td>
<td>return 0; } } → return 0; } }</td>
</tr>
<tr>
<td><strong>ID-type</strong> <em>(delete/insert/replace type)</em></td>
<td>string tmp; → char tmp;</td>
</tr>
<tr>
<td><strong>ID-typo</strong> <em>(delete/insert/replace IDentifier)</em></td>
<td>int a, b=0, m; → int a, m;</td>
</tr>
</tbody>
</table>
2. Self-supervised learning

2. How to design corruption procedure $P$?

- Look at common errors
- Design perturbation modules $M$ to cause those errors

<table>
<thead>
<tr>
<th>Perturbation module</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax (delete/insert/replace operators .();{}&quot;+, etc.)</td>
<td>return 0; } → return 0; } }</td>
</tr>
<tr>
<td>ID-type (delete/insert/replace type)</td>
<td>string tmp; → char tmp;</td>
</tr>
<tr>
<td>ID-typo (delete/insert/replace IDentifier)</td>
<td>int a, b=0, m; → int a, m;</td>
</tr>
<tr>
<td>Keyword (delete/insert/replace keyword/call)</td>
<td>if (n &gt;= 0) → while (n &gt;= 0)</td>
</tr>
</tbody>
</table>
2. Self-supervised learning

2.  Our corruption procedure $P$

- Look at common errors
- Design perturbation modules $M$ to cause those errors
- $P$: Sample 1-5 modules from $M$, and apply to program sequentially
2. Self-supervised learning

2. Our corruption procedure $P$

- Look at common errors
- Design perturbation modules $M$ to cause those errors
- $P$: Sample 1-5 modules from $M$, and apply to program sequentially
  
e.g. ID-type, ID-typo, Syntax.
2. Self-supervised learning

2. Our corruption procedure $P$
   - Look at common errors
   - Design perturbation modules $\mathcal{M}$ to cause those errors
   - $P$: Sample 1-5 modules from $\mathcal{M}$, and apply to program sequentially
e.g. ID-type, ID-typo, Syntax.

Working code
5 int i, n;
6 string A;
7 cin >> n;
8 A.resize(n);
9 for(i=0; i<n; i++){ 
10 cin >> A[i];
11 cout << i; }

2. Self-supervised learning

2. Our corruption procedure $P$
   - Look at common errors
   - Design perturbation modules $M$ to cause those errors
   - $P$: Sample 1-5 modules from $M$, and apply to program sequentially
     e.g. ID-type, ID-typo, Syntax.

Working code

<table>
<thead>
<tr>
<th>Working code</th>
<th>Perturbed 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5  int i, n;</td>
<td>5  int i, n;</td>
</tr>
<tr>
<td>6  string A;</td>
<td>6  char A;</td>
</tr>
<tr>
<td>7  cin &gt;&gt; n;</td>
<td>7  cin &gt;&gt; n;</td>
</tr>
<tr>
<td>8  A.resize(n);</td>
<td>8  A.resize(n);</td>
</tr>
<tr>
<td>9  for(i=0; i&lt;n; i++)</td>
<td></td>
</tr>
<tr>
<td>10   cin &gt;&gt; A[i];</td>
<td></td>
</tr>
<tr>
<td>11   cout &lt;&lt; i; }</td>
<td></td>
</tr>
<tr>
<td>9  for(i=0; i&lt;n; i++)</td>
<td></td>
</tr>
<tr>
<td>10   cin &gt;&gt; A[i];</td>
<td></td>
</tr>
<tr>
<td>11   cout &lt;&lt; i; }</td>
<td></td>
</tr>
</tbody>
</table>
2. Self-supervised learning

2. Our corruption procedure $P$

- Look at common errors
- Design perturbation modules $M$ to cause those errors
- $P$: Sample 1-5 modules from $M$, and apply to program sequentially
  e.g. **ID-type**, **ID-typo**, **Syntax**.

### Working code

```cpp
5  int i, n;
6  string A;
7  cin >> n;
8  A.resize(n);
9  for (i=0; i<n; i++){
10   cin >> A[i];
11   cout << i; }
```

### Perturbed 1

```cpp
6  char A;
7  cin >> n;
8  A.resize(n);
9  for (i=0; i<n; i++){
10   cin >> A[i];
11   cout << i; }
```

### Perturbed 2

```cpp
6  char A;
7  cin >> n;
8  A.resize(n);
9  for (i=0; i<n; i++){
10   cin >> A[j];
11   cout << i; }
```
2. Self-supervised learning

2. Our corruption procedure $P$
   - Look at common errors
   - Design perturbation modules $M$ to cause those errors
   - $P$: Sample 1-5 modules from $M$, and apply to program sequentially
     e.g. **ID-type**, **ID-typo**, **Syntax**.

---

**Working code**
```
5  int i, n;
6  string A;
7  cin >> n;
8  A.resize(n);
9  for(i=0; i<n; i++){
10   cin >> A[i];
11   cout << i; }
```

**Perturbed 1**
```
5  int i, n;
6  char A;
7  cin >> n;
8  A.resize(n);
9  for(i=0; i<n; i++){
10   cin >> A[j];
11   cout << i; }
```

**Perturbed 2**
```
5  int i, n;
6  char A;
7  cin >> n;
8  A.resize(n);
9  for(i=0; i<n; i++){
10   cin >> A[j];
11   cout << i; }
```

**Perturbed 3**
```
5  int i, n;
6  char A;
7  cin >> n;
8  A.resize(n);
9  for(i=0; i<n; i++){
10   cin >> A[j];
11   cout << i; }
```
2. Self-supervised learning

3. Prepare pre-training data
   ● 300K working programs from codeforces.com

```cpp
5 int i, n;
6 string A;
7 cin >> n;
8 A.resize(n);
9 for (i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```
2. Self-supervised learning

3. Prepare pre-training data
   - 300K working programs from codeforces.com
   - For each program, create corrupted versions by applying $P$

```cpp
5  int i, n;
6  string A;
7  cin >> n;
8  A.resize(n);
9  for (i=0;i<n;i++){
10   cin >> A[i];
11   cout << i; }
```

```
5  int i, n;
6  char A;
7  cin >> n
8  A.resize(n);
9  for (i=0;i<n;i++){
10   cin >> A[j];
11   cout << i; }
```
2. Self-supervised learning

3. Prepare pre-training data
   ● 300K working programs from codeforces.com
   ● For each program, create corrupted versions by applying $P$

```
Working code
5    int i, n;
6    string A;
7    cin >> n;
8    A.resize(n);
9    for (i=0;i<n;i++){
10       cin >> A[i];
11       cout << i; }

Corrupted
5    int i, n;
6    char A;
7    cin >> n
8    A.resize(n);
9    for (i=0;i<n;i++){
10       cin >> A[j];
11       cout << i; }
```

Error!
line 7: expected ‘;’
2. Self-supervised learning

3. Prepare pre-training data
   - 300K working programs from codeforces.com
   - For each program, create corrupted versions by applying $P$
     ⇒ New program repair examples: <broken code, feedback, fixed code>
2. Self-supervised learning

3. Prepare pre-training data
- 300K working programs from codeforces.com
- For each program, create corrupted versions by applying $P$
  $\Rightarrow$ New program repair examples: <broken code, feedback, fixed code>
2. Self-supervised learning

3. Prepare pre-training data
   - 300K working programs from codeforces.com
   - For each program, create corrupted versions by applying $P$
     $\Rightarrow$ New program repair examples: <broken code, feedback, fixed code>

```
int i, n;
string A;
cin >> n;
A.resize(n);
for (i=0;i<n;i++){
cin >> A[i];
cout << i; }
```

Error!
line 7: expected ‘;’
2. Self-supervised learning

3. Prepare pre-training data

- 300K working programs from codeforces.com
- For each program, create corrupted versions by applying $P$

$\Rightarrow$ New program repair examples: <broken code, feedback, fixed code>
2. Self-supervised learning

What’s interesting?

- Typically, pre-training task ≠ target task (e.g. masked LM v.s. QA)
- Here, targeted pre-training (pre-training task = target task = program repair)
  - More direct pre-training structure
  - Data distributions can be different between pre-training & target
Evaluation 1: DeepFix
Evaluation 1: DeepFix

Task

- Repair C programs
- May have **multiple error lines**
- Apply repair model iteratively (up to 5 times)

```c
#include <stdio.h>
#include <stdlib.h>
int pow(int a, int b);
int main(){
    int n;
    scanf("%d",&n);
    int i, j;
    for(i=1;i<=n;i++){
        for(j=0;j<=n;j++){
            if(j<i){
                printf("%d ",pow(i,j));}
            printf("\n");
        return 0;
    }
```
Evaluation 1: DeepFix

Our model outputs

Input code

4 int main() {
5    int n;
6    int * m[2];
7    m[0] = malloc(n*sizeof(int));
8    m[1] = malloc(n*sizeof(int));
9    for (i = 0; i < n; i++) {
10       m[0][i] = -1;
11       m[1][i] = -1; }
12    return 0 }

Evaluation 1: DeepFix

Our model outputs

Input code

4 int main() {
5   int n;
6   int * m[2];
7   m[0] = malloc(n*sizeof(int));
8   m[1] = malloc(n*sizeof(int));
9   for (i = 0; i < n; i++) {
10      m[0][i] = -1;
11      m[1][i] = -1; }
12   return 0 }

Error message

line 9: ‘i’ undeclared
Evaluation 1: DeepFix

Our model outputs

Input code

```c
4 int main() { 
5  int n;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Attempt 1

```c
4 int main() {
5  int n, i;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

Error message

line 9: ‘i’ undeclared
Evaluation 1: DeepFix

Our model outputs

**Input code**

```c
4 int main() {
5  int n;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

**Attempt 1**

```c
4 int main() {
5  int n, i;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10   m[0][i] = -1;
11   m[1][i] = -1; }
12  return 0 }
```

**Error message**

- line 9: ‘i’ undeclared

**Error message**

- line 12: expected ‘;’ before ‘}’
Evaluation 1: DeepFix

Our model outputs

Input code

4 int main() {
5  int n;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10     m[0][i] = -1;
11     m[1][i] = -1;
12  return 0; }

Error message

line 9: ‘i’ undeclared

Attempt 1

4 int main() {
5  int n, i;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10     m[0][i] = -1;
11     m[1][i] = -1;
12     return 0; }

Error message

line 12: expected ‘;’ before ‘}’

Attempt 2

4 int main() {
5  int n;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10     m[0][i] = -1;
11     m[1][i] = -1;
12     return 0; }
Evaluation 1: DeepFix

Our model outputs

Input code

4 int main() {
5  int n;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10     m[0][i] = -1;
11     m[1][i] = -1; }
12  return 0 }

Error message
line 9: ‘i’ undeclared

DrRepair

Attempt 1

4 int main() {
5  int n, i;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10     m[0][i] = -1;
11     m[1][i] = -1; }
12  return 0 }

Error message
line 12: expected ‘;’ before ‘}’

DrRepair

Attempt 2

4 int main() {
5  int n;
6  int * m[2];
7  m[0] = malloc(n*sizeof(int));
8  m[1] = malloc(n*sizeof(int));
9  for (i = 0; i < n; i++) {
10     m[0][i] = -1;
11     m[1][i] = -1; }
12  return 0; }

Compiled!!
Evaluation 1: DeepFix

Results

Test (full repair accuracy)

Prior works do not use compiler messages
Evaluation 1: DeepFix

Results

Test (full repair accuracy)

Prior works do not use compiler messages

Gupta+17: 27.0%
Gupta+19: 26.6%
Hajipour+19: 45.3%
Ours: no compiler 34.0%
Evaluation 1: DeepFix

Results

Test (full repair accuracy)

Use of compiler messages is important

Prior works do not use compiler messages

Gupta+17 27.0%  Gupta+19 26.6%  Hajipour+19 45.3%  Ours: no compiler 34.0%  Ours: base 62.5%
Evaluation 1: DeepFix

Results

Test (full repair accuracy)

Use of **compiler messages** is important

Prior works do not use compiler messages

<table>
<thead>
<tr>
<th>System</th>
<th>Gupta+17</th>
<th>Gupta+19</th>
<th>Hajipour+19</th>
<th>Ours: no compiler</th>
<th>Ours: base</th>
<th>Ours: base + graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy (%)</td>
<td>27.0%</td>
<td>26.6%</td>
<td>45.3%</td>
<td>34.0%</td>
<td>62.5%</td>
<td>66.4%</td>
</tr>
</tbody>
</table>
Evaluation 1: DeepFix

Results

Test (full repair accuracy)

Prior works do not use compiler messages

Use of compiler messages is important

Graph & pre-train are both useful

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gupta+17</td>
<td>27.0%</td>
</tr>
<tr>
<td>Gupta+19</td>
<td>26.6%</td>
</tr>
<tr>
<td>Hajipour+19</td>
<td>45.3%</td>
</tr>
<tr>
<td>Our: no compiler</td>
<td>34.0%</td>
</tr>
<tr>
<td>Ours: base</td>
<td>62.5%</td>
</tr>
<tr>
<td>Ours: base + graph</td>
<td>66.4%</td>
</tr>
<tr>
<td>Ours: base + graph + pretrain (DrRepair)</td>
<td>68.2%</td>
</tr>
</tbody>
</table>
Evaluation 2: SPoC
Evaluation 2: SPoC

Task

- Translate pseudocode into C++ code (program synthesis)
- Line-level alignment

<table>
<thead>
<tr>
<th>i</th>
<th>( x_i )</th>
<th>( y_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>in function main</td>
<td>int main() {</td>
</tr>
<tr>
<td>2</td>
<td>let n be integer</td>
<td>int n;</td>
</tr>
<tr>
<td>3</td>
<td>read n</td>
<td>cin &gt;&gt; n;</td>
</tr>
<tr>
<td>4</td>
<td>let A be vector of integers</td>
<td>vector&lt;int&gt; A;</td>
</tr>
<tr>
<td>5</td>
<td>set size of A = n</td>
<td>A.resize(n);</td>
</tr>
<tr>
<td>6</td>
<td>read n elements into A</td>
<td>for(int i = 0; i &lt; A.size(); i++) cin &gt;&gt; A[i];</td>
</tr>
<tr>
<td>7</td>
<td>for all elements in A</td>
<td>for(int i = 0; i &lt; A.size(); i++) {</td>
</tr>
<tr>
<td>8</td>
<td>set min_i to i</td>
<td>int min_i = i;</td>
</tr>
</tbody>
</table>

[Kulal et al., 19]
Evaluation 2: SPoC

Prior work (Kulal+19)
- Stitch line-level translations & search

Pseudocode

```
declare i
for i from 0 to n
print i
```
Evaluation 2: SPoC

Prior work (Kulal+19)
- Stitch line-level translations & search

**Pseudocode**

```
declare i
for i from 0 to n
print i
```

**Code candidates**

```
int i;               char i;           i=0;
[p=0.7]              [p=0.2]                  [p=0.1]
```

`translate`
Evaluation 2: SPoC

Prior work (Kulal+19)

- Stitch line-level translations & search

**Pseudocode**

```
declare i
for i from 0 to n
print i
```

**Code candidates**

<table>
<thead>
<tr>
<th>int i;</th>
<th>char i;</th>
<th>i=0;</th>
</tr>
</thead>
<tbody>
<tr>
<td>![p=0.7]</td>
<td>![p=0.2]</td>
<td>![p=0.1]</td>
</tr>
<tr>
<td>for(int i=0;i&lt;n;i++)</td>
<td>for(i=0;i&lt;n;i++)</td>
<td>while (1)</td>
</tr>
<tr>
<td>![p=0.3]</td>
<td>![p=0.2]</td>
<td>![p=0.1]</td>
</tr>
</tbody>
</table>
### Evaluation 2: SPoC

**Prior work** (Kulal+19)
- Stitch line-level translations & search

#### Pseudocode

```plaintext
declare i
for i from 0 to n
print i
```

#### Code candidates

<table>
<thead>
<tr>
<th>Code candidates</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i;</td>
<td>p=0.7</td>
</tr>
<tr>
<td>char i;</td>
<td>p=0.2</td>
</tr>
<tr>
<td>i=0;</td>
<td>p=0.1</td>
</tr>
<tr>
<td>for(int i=0;i&lt;n;i++)</td>
<td>p=0.3</td>
</tr>
<tr>
<td>for(i=0;i&lt;n;i++)</td>
<td>p=0.2</td>
</tr>
<tr>
<td>while (1)</td>
<td>p=0.1</td>
</tr>
<tr>
<td>cout &lt;&lt; i &lt;&lt; &quot;\n&quot;;</td>
<td>p=0.4</td>
</tr>
<tr>
<td>cout &lt;&lt; i;</td>
<td>p=0.3</td>
</tr>
<tr>
<td>putc(i);</td>
<td>p=0.1</td>
</tr>
</tbody>
</table>
Evaluation 2: SPoC

Prior work (Kulal+19)
- Stitch line-level translations & search

**Pseudocode**
- declare i
- for i from 0 to n
- print i

**Code candidates**
- int i;
- char i;
- i=0;
- for(int i=0;i<n;i++)
- for(i=0;i<n;i++)
- while (1)
- cout << i << "\n";
- cout << i;
- putc(i);
Evaluation 2: SPoC

Prior work (Kulal+19)
- Stitch line-level translations & search

Pseudocode
- declare i
- for i from 0 to n
- print i

Code candidates
- int i; [p=0.7]
- for(int i=0;i<n;i++) [p=0.3]
- cout << i << "\n"; [p=0.4]

- char i; [p=0.2]
- for(i=0;i<n;i++) while (1) [p=0.2]
- cout << i; [p=0.3]

- i=0; [p=0.1]
- for(i=0;i<n;i++) [p=0.1]
- putc(i); [p=0.1]

Best first search
Evaluation 2: SPoC

Prior work (Kulal+19)

- Stitch line-level translations & search

**Pseudocode**

```
declare i
for i from 0 to n
print i
```

**Code candidates**

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>C++ Code 1</th>
<th>C++ Code 2</th>
<th>C++ Code 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare i</td>
<td>int i;</td>
<td>char i;</td>
<td>i=0;</td>
</tr>
<tr>
<td>for i from 0 to n</td>
<td>for(int i=0;i&lt;n;i++)</td>
<td>for(i=0;i&lt;n;i++) while (1)</td>
<td></td>
</tr>
<tr>
<td>print i</td>
<td>cout &lt;&lt; i &lt;&lt; &quot;\n&quot;;</td>
<td>cout &lt;&lt; i;</td>
<td>putc(i);</td>
</tr>
</tbody>
</table>

Best first search
Evaluation 2: SPoC

Prior work (Kulal+19)
- Stitch line-level translations & search

Pseudocode

```plaintext
declare i
for i from 0 to n
print i
```

Code candidates

```plaintext
int i;
for(int i=0;i<n;i++)
cout << i << "\n";
```

- `int i;` [p=0.7]
- `for(int i=0;i<n;i++)` [p=0.3]
- `cout << i << "\n";` [p=0.4]

```plaintext
char i;
for(i=0;i<n;i++)
cout << i;
```

- `char i;` [p=0.2]
- `for(i=0;i<n;i++)` [p=0.2]
- `cout << i;` [p=0.3]

```plaintext
i=0;
while (1)
putc(i);
```

- `i=0;` [p=0.1]
- `while (1)` [p=0.1]
- `putc(i);` [p=0.1]

Best first search ...
Evaluation 2: SPoC

Problem
- Line-level translation misses global context

Code candidates

<table>
<thead>
<tr>
<th>Language</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i;</td>
<td>[p=0.7]</td>
</tr>
<tr>
<td>char i;</td>
<td>[p=0.2]</td>
</tr>
<tr>
<td>i=0;</td>
<td>[p=0.1]</td>
</tr>
<tr>
<td>for(int i=0;i&lt;n;i++)</td>
<td>[p=0.3]</td>
</tr>
<tr>
<td>for(i=0;i&lt;n;i++)</td>
<td>[p=0.2]</td>
</tr>
<tr>
<td>while (1)</td>
<td>[p=0.1]</td>
</tr>
<tr>
<td>cout &lt;&lt; i &lt;&lt; &quot;\n&quot;;</td>
<td>[p=0.4]</td>
</tr>
<tr>
<td>cout &lt;&lt; i;</td>
<td>[p=0.3]</td>
</tr>
<tr>
<td>putc(i);</td>
<td>[p=0.1]</td>
</tr>
</tbody>
</table>
Evaluation 2: SPoC

Problem

- Line-level translation misses global context

**Code candidates**

<table>
<thead>
<tr>
<th>Probability</th>
<th>Code</th>
<th>Does not compile!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>int i;</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>for(int i=0;i&lt;n;i++)</td>
<td>(redeclaration of ‘i’)</td>
</tr>
<tr>
<td>0.4</td>
<td>cout &lt;&lt; i &lt;&lt; &quot;\n&quot;;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Probability</th>
<th>Code</th>
<th>Does not compile!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>char i;</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>for(i=0;i&lt;n;i++)</td>
<td>(redeclaration of ‘i’)</td>
</tr>
<tr>
<td>0.3</td>
<td>cout &lt;&lt; i;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Probability</th>
<th>Code</th>
<th>Does not compile!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>i=0;</td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>while (1)</td>
<td>(redeclaration of ‘i’)</td>
</tr>
<tr>
<td>0.1</td>
<td>putc(i);</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation 2: SPoC

Problem
- Line-level translation misses global context

<table>
<thead>
<tr>
<th>Code candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>int i;</td>
</tr>
<tr>
<td>char i;</td>
</tr>
<tr>
<td>i=0;</td>
</tr>
<tr>
<td>for(int i=0;i&lt;n;i++)</td>
</tr>
<tr>
<td>for(i=0;i&lt;n;i++)</td>
</tr>
<tr>
<td>while (1)</td>
</tr>
<tr>
<td>cout &lt;&lt; i &lt;&lt; &quot;\n&quot;;</td>
</tr>
<tr>
<td>cout &lt;&lt; i;</td>
</tr>
<tr>
<td>putc(i);</td>
</tr>
</tbody>
</table>

[p=0.7] [p=0.2] [p=0.1]
[p=0.3] [p=0.2] [p=0.1]
[p=0.4] [p=0.3] [p=0.1]

Does not compile!
(redeclaration of ‘i’)

Our solution
- Apply repair model if current candidate program does not compile
Evaluation 2: SPoC

Results

TestP (synthesis success rate)

- Top1 stitch: 17.8%
- Best first search: 32.5%
- Kulal+19: 34.2%

BFS + error localization
Evaluation 2: SPoC

Results

TestP (synthesis success rate)

BFS + error localization

37.6%

Ours (DrRepair)

Top1 stitch

Best first search

Kulal+19

17.8%

32.5%

34.2%
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation

- Our base: 48.6%

TestP (synthesis success rate)

- Top1 stitch: 17.8%
- Best first search: 32.5%
- Kulal+19: 34.2%
- Ours (DrRepair): 37.6%
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation

TestP (synthesis success rate)

- BFS + error localization
- BFS + repair

Ours (DrRepair)

- Top1 stitch: 17.8%
- Best first search: 32.5%
- Kulal+19: 34.2%
- Ours: 37.6%
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation

TestP (synthesis success rate)

- BFS + error localization
- BFS + repair

- Top1 stitch
- Best first search
- Kulal+19
- Ours (DrRepair)

<table>
<thead>
<tr>
<th></th>
<th>Ours (DrRepair)</th>
<th>TestP (synthesis success rate)</th>
<th>Dev (repair accuracy) - ablation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our base</td>
<td>48.6%</td>
<td>17.8%</td>
<td>48.6%</td>
</tr>
<tr>
<td>Our base + graph</td>
<td>53.0%</td>
<td>32.5%</td>
<td>53.0%</td>
</tr>
<tr>
<td>Our base + graph + pretrain (DrRepair)</td>
<td>56.2%</td>
<td>37.6%</td>
<td>56.2%</td>
</tr>
</tbody>
</table>
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our base</td>
<td>48.6%</td>
</tr>
<tr>
<td>Our base + graph</td>
<td>53.0%</td>
</tr>
<tr>
<td>Our base + graph + pretrain (DrRepair)</td>
<td>56.2%</td>
</tr>
<tr>
<td>Our base + pseudocode</td>
<td>66.2%</td>
</tr>
</tbody>
</table>

TestP (synthesis success rate)

<table>
<thead>
<tr>
<th>Method</th>
<th>Success Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top1 stitch</td>
<td>17.8%</td>
</tr>
<tr>
<td>Best first search</td>
<td>32.5%</td>
</tr>
<tr>
<td>Kulal+19</td>
<td>34.2%</td>
</tr>
<tr>
<td>Ours (DrRepair)</td>
<td>37.6%</td>
</tr>
</tbody>
</table>

BFS + error localization

BFS + repair
Evaluation 2: SPoC

Results

Dev (repair accuracy) - ablation

TestP (synthesis success rate)
Analysis 1: When is `graph` useful?

<table>
<thead>
<tr>
<th>Compiler message type</th>
<th>Frequency in train set (SPoC)</th>
<th>Repair acc. (SPoC dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>base</td>
</tr>
<tr>
<td>‘@@@’ was not declared …</td>
<td>35.2 %</td>
<td>50.2</td>
</tr>
<tr>
<td>redeclaration of ‘@@@’</td>
<td>8.9 %</td>
<td>40.7</td>
</tr>
<tr>
<td>expected ‘@@@’ before ‘@@@’</td>
<td>3.2 %</td>
<td>67.6</td>
</tr>
<tr>
<td>expected primary-expression before …</td>
<td>3.0 %</td>
<td>47.4</td>
</tr>
<tr>
<td>request for member ‘@@@’ in ‘@@@’, …</td>
<td>2.9 %</td>
<td>37.9</td>
</tr>
<tr>
<td>expected initializer before ‘@@@’</td>
<td>2.1 %</td>
<td>48.8</td>
</tr>
<tr>
<td>‘@@@’ without a previous ‘@@@’</td>
<td>1.3 %</td>
<td>37.0</td>
</tr>
</tbody>
</table>
Analysis 1: When is **graph** useful?

<table>
<thead>
<tr>
<th>Compiler message type</th>
<th>Frequency in train set (SPoC)</th>
<th>Repair acc. (SPoC dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>base</td>
</tr>
<tr>
<td>‘@@@’ was not declared …</td>
<td>35.2 %</td>
<td>50.2</td>
</tr>
<tr>
<td>redeclaration of ‘@@@’</td>
<td>8.9 %</td>
<td>40.7</td>
</tr>
<tr>
<td>expected ‘@@@’ before ‘@@@’</td>
<td>3.2 %</td>
<td>67.6</td>
</tr>
<tr>
<td>expected primary-expression before …</td>
<td>3.0 %</td>
<td>47.4</td>
</tr>
<tr>
<td>request for member ‘@@@’ in ‘@@@’, …</td>
<td>2.9 %</td>
<td>37.9</td>
</tr>
<tr>
<td>expected initializer before ‘@@@’</td>
<td>2.1 %</td>
<td>48.8</td>
</tr>
<tr>
<td>‘@@@’ without a previous ‘@@@’</td>
<td>1.3 %</td>
<td>37.0</td>
</tr>
</tbody>
</table>
Analysis 1: When is **graph** useful?

<table>
<thead>
<tr>
<th>Compiler message type</th>
<th>Frequency in train set (SPoC)</th>
<th>Repair acc. (SPoC dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>base</td>
<td>+ graph</td>
</tr>
<tr>
<td>‘@@@’ was not declared ...</td>
<td>35.2 %</td>
<td>50.2</td>
</tr>
<tr>
<td>redeclaration of ‘@@@’</td>
<td>8.9 %</td>
<td>40.7</td>
</tr>
<tr>
<td>expected ‘@@@’ before ‘@@@’</td>
<td>3.2 %</td>
<td>67.6</td>
</tr>
<tr>
<td>expected primary-expression before ...</td>
<td>3.0 %</td>
<td>47.4</td>
</tr>
<tr>
<td>request for member ‘@@@’ in ‘@@@’, ...</td>
<td>2.9 %</td>
<td>37.9</td>
</tr>
<tr>
<td>expected initializer before ‘@@@’</td>
<td>2.1 %</td>
<td>48.8</td>
</tr>
<tr>
<td>‘@@@’ without a previous ‘@@@’</td>
<td>1.3 %</td>
<td>37.0</td>
</tr>
</tbody>
</table>

Need reasoning over multiple lines of code
Recall this example

Broken program

```cpp
#include <bits/stdc++.h>
#include <string>
using namespace std;
int main() {
    char tmp, a, b;
    map<string, int> mp;
    cin >> a >> b;
    int i, j;
    for (i = 0; i < a.size(); i++)
        tmp.push_back(a[i]);
    string tmp1 = tmp;
    for (j = 0; j < b.size(); j++)
        tmp1.push_back(b[j]);
    mp[tmp1] = 1;
    }
    }
```

Feedback

- line 9: error: request for member ‘size’ in ‘a’, which is of non-class type ‘char’

Repair

1. Error localized line 5
2. Repair

```cpp
char tmp, a, b;
→ string tmp, a, b;
```
Analysis 1: When is graph useful?

<table>
<thead>
<tr>
<th>Compiler message type</th>
<th>Frequency in train set (SPoC)</th>
<th>Repair acc. (SPoC dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>base</td>
</tr>
<tr>
<td>‘@@@’ was not declared</td>
<td>35.2 %</td>
<td>50.2</td>
</tr>
<tr>
<td>redeclaration of ‘@@@’</td>
<td>8.9 %</td>
<td>40.7</td>
</tr>
<tr>
<td>expected ‘@@@’ before ‘@@@’</td>
<td>3.2 %</td>
<td>67.6</td>
</tr>
<tr>
<td>expected primary-expression before ...</td>
<td>3.0 %</td>
<td>47.4</td>
</tr>
<tr>
<td>request for member ‘@@@’ in ‘@@@’, ...</td>
<td>2.9 %</td>
<td>37.9</td>
</tr>
<tr>
<td>expected initializer before ‘@@@’</td>
<td>2.1 %</td>
<td>48.8</td>
</tr>
<tr>
<td>‘@@@’ without a previous ‘@@@’</td>
<td>1.3 %</td>
<td>37.0</td>
</tr>
</tbody>
</table>
## Analysis 2: When is **pre-training** useful?

<table>
<thead>
<tr>
<th>Compiler message type</th>
<th>Frequency in train set (SPoC)</th>
<th>Repair acc. (SPoC dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>base</td>
</tr>
<tr>
<td>‘@@@’ was not declared ...</td>
<td>35.2 %</td>
<td>50.2</td>
</tr>
<tr>
<td>redeclaration of ‘@@@’</td>
<td>8.9 %</td>
<td>40.7</td>
</tr>
<tr>
<td>expected ‘@@@’ before ‘@@@’</td>
<td>3.2 %</td>
<td>67.6</td>
</tr>
<tr>
<td>expected primary-expression before ...</td>
<td>3.0 %</td>
<td>47.4</td>
</tr>
<tr>
<td>request for member ‘@@@’ in ‘@@@’, ...</td>
<td>2.9 %</td>
<td>37.9</td>
</tr>
<tr>
<td>expected initializer before ‘@@@’</td>
<td>2.1 %</td>
<td>48.8</td>
</tr>
<tr>
<td>‘@@@’ without a previous ‘@@@’</td>
<td>1.3 %</td>
<td>37.0</td>
</tr>
</tbody>
</table>
Analysis 2: When is **pre-training** useful?

<table>
<thead>
<tr>
<th>Compiler message type</th>
<th>Frequency in train set (SPoC)</th>
<th>Repair acc. (SPoC dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>base</td>
<td>+ graph</td>
</tr>
<tr>
<td>‘@@@’ was not declared ...</td>
<td>35.2 %</td>
<td>50.2</td>
</tr>
<tr>
<td>redeclaration of ‘@@@’</td>
<td>8.9 %</td>
<td>40.7</td>
</tr>
<tr>
<td>expected ‘@@@’ before ‘@@@’</td>
<td>3.2 %</td>
<td>67.6</td>
</tr>
<tr>
<td>expected primary-expression before ...</td>
<td>3.0 %</td>
<td>47.4</td>
</tr>
<tr>
<td>request for member ‘@@@’ in ‘@@@’, ...</td>
<td>2.9 %</td>
<td>37.9</td>
</tr>
<tr>
<td>expected initializer before ‘@@@’</td>
<td>2.1 %</td>
<td>48.8</td>
</tr>
<tr>
<td>‘@@@’ without a previous ‘@@@’</td>
<td>1.3 %</td>
<td>37.0</td>
</tr>
</tbody>
</table>

*Rare in train set*
*Extra examples in pre-training are helpful*
Takeaways

New insights

● Use of **error messages** is crucial to learn program repair
● Program-feedback **graph** helps complex reasoning
● Unlabeled programs can be used for **targeted pre-training**

General framework

● **Learning to reason with feedback** - many applications, e.g.
  ○ Edit essays based on written feedback
  ○ Learn from user inputs in interactive dialogue
● We show that **graph-based reasoning** can be a good solution
Thanks!

Thank you to John Hewitt, Mina Lee, Sumith Kulal, Pang Wei Koh, Robin Jia, Ananya Kumar, Ruiqi Zhong, and anonymous reviewers.

Funded in part by NSF CAREER Award, PECASE Award, and Amazon Research Award.

Code/Data: [https://cs.stanford.edu/~myasu/](https://cs.stanford.edu/~myasu/)